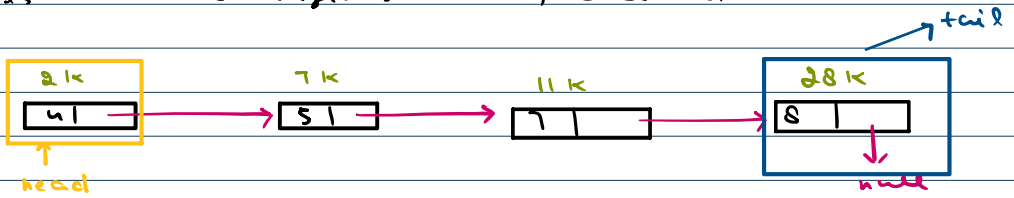
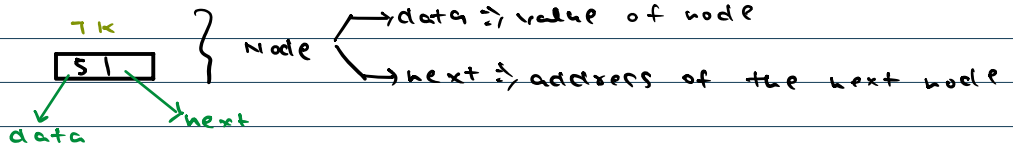
Memory → Contagious allocation
Memory → Non contagious allocation

Linked list

It is a linear data structure.
where elements (called nodes) are connected through references.
It is a non-contagious memory allocation.

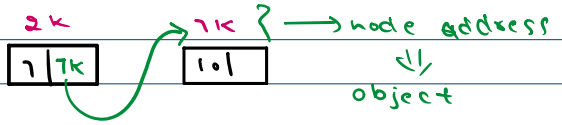




Characteristics

1. Non-contigious memory allocation
2. Dynamic memory allocation
3. No element shifting (insert/delete)
4. Sequential access
5. Extra memory overhead

```
class Node {
    int data;
    Node next;
}
```

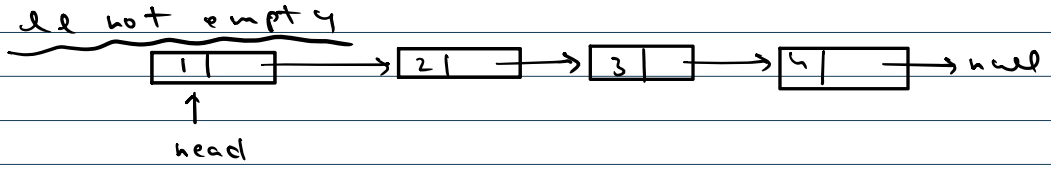


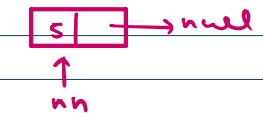
Linked List operations → add, remove, set

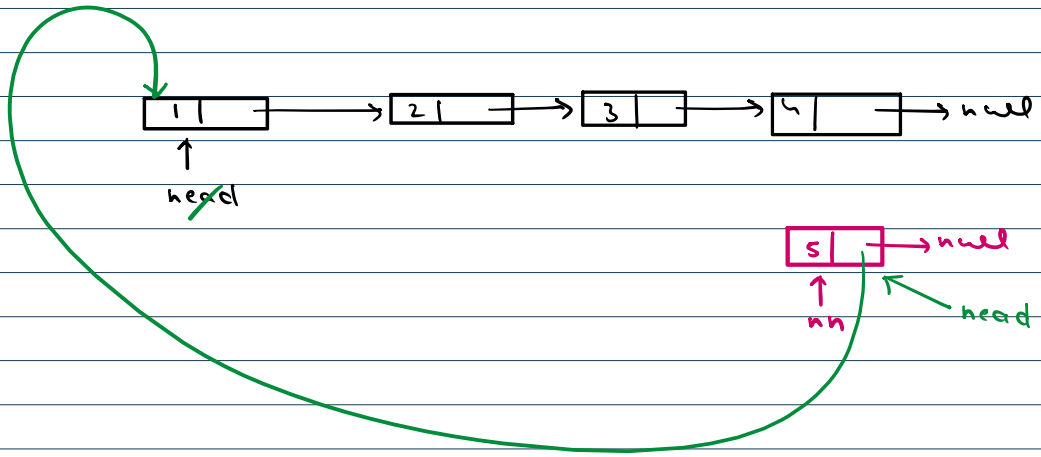
add first → if empty

addFirst → ll empty
→ ll not empty

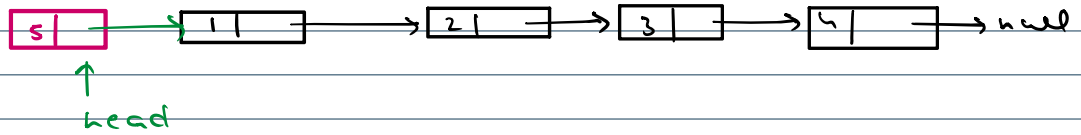
ll not empty





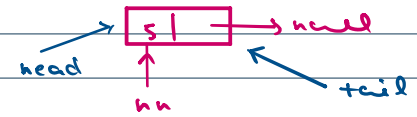


nn.next = head
head = nn



ll empty

head : null
tail : null

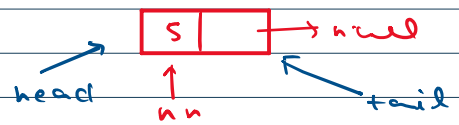


head = nn
tail = nn

addLast → ll empty
→ ll not empty

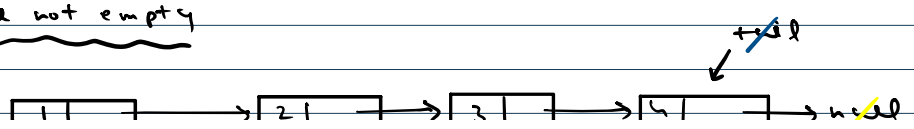
ll empty

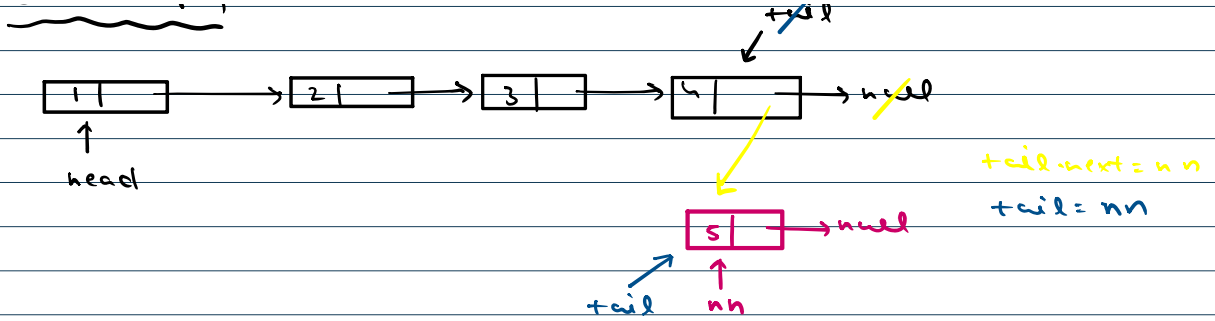
head : null
tail : null



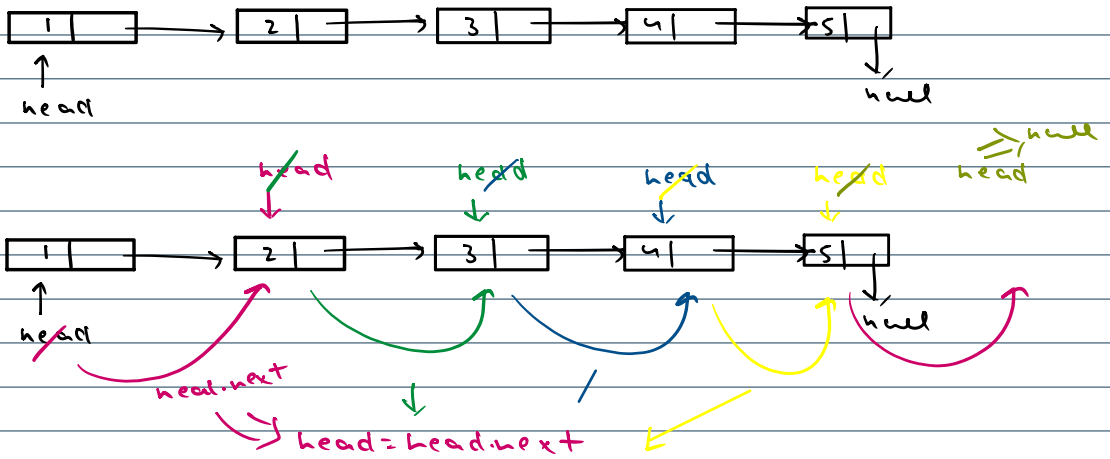
head = nn
tail = nn

ll not empty





display ll



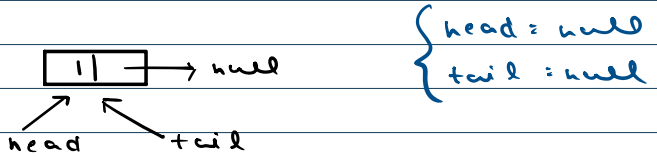

```
while(head != null ) {
        print (head.val)
        head = head.next
}
```

remove

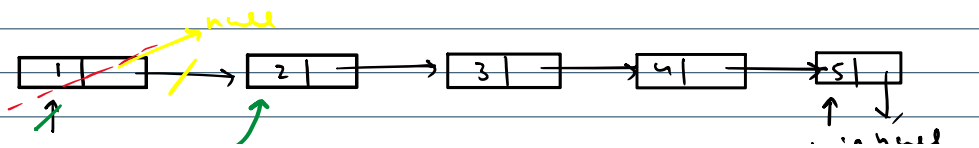
remove first

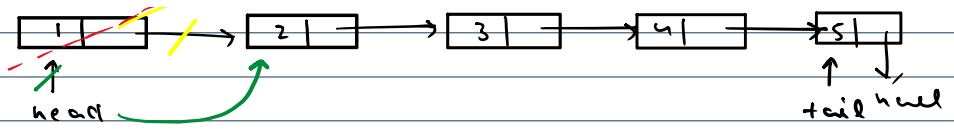
- ll > 1
- ll <= 1

ll <= 1



ll > 1





temp = head

head = head.next

temp.next = null

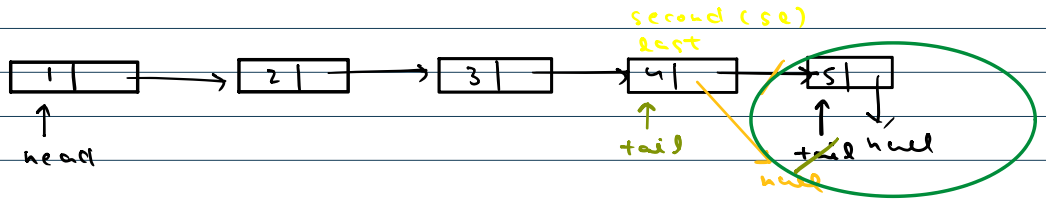
remove Last

ll <= 1

ll > 1

ll <= 1 => removeFirst() => tail = null
head = null

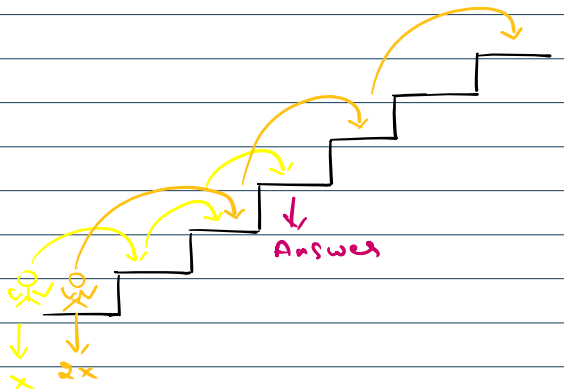
ll > 1



tail = sl

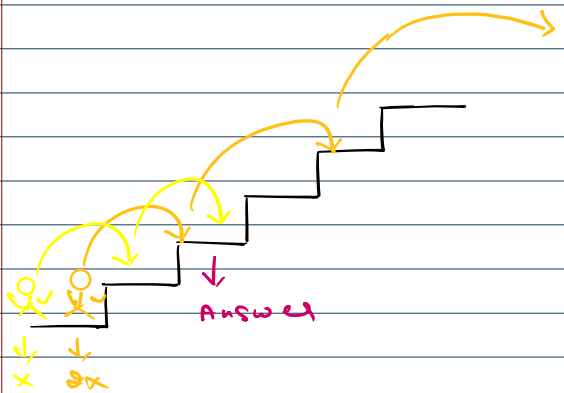
sl.next = null

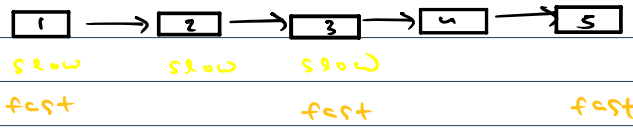
middle of the linked list

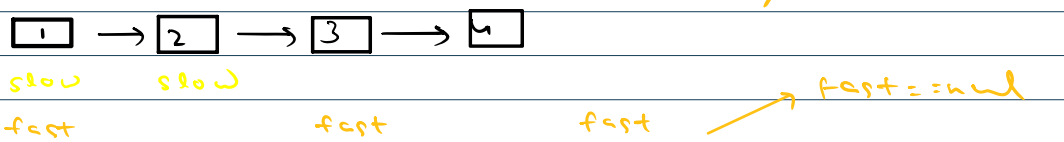


=> Abhay = 2x => in 3 steps each end

=> Aarush = x => in 3 steps at middle

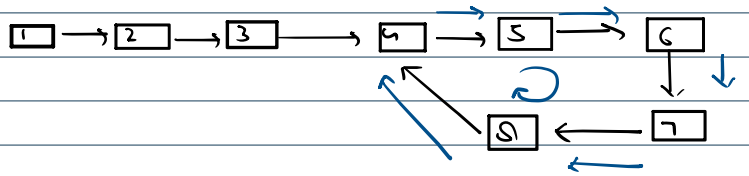


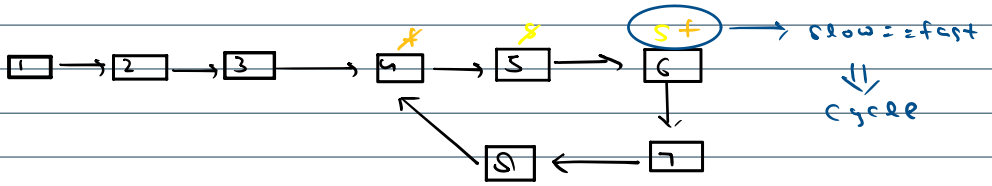




```
class Solution {
    public ListNode middleNode(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
}
```

Linked List Cycle





```
public class Solution {
    public boolean hasCycle(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        while(fast != null && fast.next != null){
            fast = fast.next.next;
            slow = slow.next;
            if(slow == fast){
                return true;
            }
        }
        return false;
    }
}
```